SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ALUSTARIIPPUMATON MOBIILISOVELLUS PAIKANTAMISELLA, PAIKANNUSHISTORIALLA JA SISÄLLÖN JAKAMISELLA

TEKIJÄ: Juha Tirkkonen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Juha Tirkkonen	
Työn nimi Alustariippumaton mobiilisovellus paikantamisella, paikannushistorialla ja sisällön jakamisella	
Päiväys 5.6.2020	Sivumäärä/Liitteet 32/0
Ohjaaja(t) Ville Berg	
Toimeksiantaja/Yhteistyökumppani(t) Tatu Myöhänen	

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutkia nykyisten paikannustekniikoiden mahdollisuuksia mobiilisovelluskehityksen näkökannalta ja kehittää soveltuvuusselvitys toimeksiantajan idealle mobiilisovelluksen muodossa. Kyseessä on paikannukseen pohjautuva sovellus, jonka käyttäjät voivat kommunikoida keskenään reaaliaikaiseen sijaintiin ja sijaintihistoriaan pohjautuen.

Kyseessä on alustariippumaton sovellus, jonka tavoitteena on toimia samalla tavalla sekä Android että iOS-käyttöjärjestelmillä. Kehitystekniikkana käytettiin React Nativea ja siihen pohjautuvaa Expo ohjelmistokehityspakettia, jonka avulla molemmille käyttöjärjestelmille saatiin tehtyä sovellus lähes yhden toteutuksen työmäärällä.
Muita käytettyjä tekniikoita on React Native Maps sijainnin näyttämiseen karttanäkymänä ja tietokantoina käytettiin pilvipohjaista FireBase Cloud Firestorea, NoSQL-tyyppistä tietokantaa käyttäjien välisten tietojen jakamiseen ja SQLiteä, relaatiopohjaista ja paikallista tietokantaa yksityisen sijaintihistorian tallentamiseen.

Opinnäytetyön lopputuloksena on toimeksiantajan tärkeimpiä kriteereitä vastaava sovellus, joka toimii molemmilla käyttöjärjestelmillä. Kehitystyö onnistui React Nativella ja Expolla hyvin, haasteina oli joidenkin komponenttikirjastojen tuen puute Expolle, joka toi muutoksia suunnitelmiin käytettävien tekniikoiden suhteen.

Avainsanat
React Native, Expo, Firebase, Maps

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Juha Tirkkonen			
Title of Thesis Cross-platform mobile application with location tracking, location history and content sharing			
Date	5 June 2020	Pages/Appendices	32/0
Supervisor(s) Mr Ville Berg, Lecturer			
Client Organisation /Partners Tatu Myöhänen			

Abstract

The purpose of this thesis was to research possibilities of modern location tracking technologies in the context of mobile application development and make proof of concept for the original idea of the commissioner in the form of a mobile application. It is a location tracking based application, in which users can communicate with each other based on the real-time location and location history.

It is a cross-platform application, the aim of which is to operate in a similar manner in both Android and iOS operating systems. The main development technologies used were React Native and Expo software development kit. With Expo, it is possible to develop an application to both operating systems with almost the amount of the work of single implementation. Other technologies used were React Native Maps for showing the user location in map view, FireBase Cloud FireStore NoSQL database for storing user communication related data and local SQLite relational database for saving private location history.

The result of this thesis was an application that meets the criterion of the commissioner. Development was successful with React Native and Expo although there were some challenges. For example, some component libraries were missing official support for Expo and that brought changes to the original plans in using some of the technologies.

Keywords
React Native, Expo, Firebase, Maps

SISÄLTÖ

1 JOHDANTO

Työn tarkoituksena oli luoda alustariippumaton mobiilisovellus toimeksiantajan ideaan pohjautuvien vaatimuksien mukaan. Tavoitteena oli sovellus, jonka ominaisuuksina olisi paikantaminen ja sijaintihistoria sekä niiden näyttäminen kartalta. Sovelluksessa tulisi olla käyttäjien välistä sisällönjakamista sijaintiin perustuen. Sisällönjakaminen tarkoittaa tässä tapauksessa käyttäjien välistä kommunikointia perustuen reaaliaikaiseen sijaintiin ja sijaintihistoriaan. Kommunikointi pohjautuu tietyn ajan ja matkan etäisyyksien määrittämiin ”kohtaamisiin”, jolloin käyttäjät voivat lähettää kaveripyyntöjä tai perustaa ryhmiä ja kommunikoida lisättyjen kaverien tai ryhmien kanssa. Alkuperäinen idea kohtaamisista ovat julkiset tapahtumat kuten festarit tai vastaavasti muut, yksityiset tapaamiset tai kokoontumiset.

Toimintojen toteuttamisen lisäksi tärkeä kriteeri sovellukselle oli kustannustehokkuus. Tämä toi haasteita mm. sijaintihistorian tallentamiseen, joka teoreettisena maksimiskenaariona olisi lähes mahdotonta toteuttaa ainakin Firestoren avulla lopulliseen kaupalliseen versioon lähinnä pelkkien tietokantaan kohdistuvien siirtokustannuksien takia. Eri tarjoajien pilvipalvelu tietokantoja on hyvin hankalaa vertailla kustannusmielessä erilaisten maksukategorioiden ja hinnoitteluohjelmien vuoksi. Ratkaisuna kokonaisen sijaintihistorian tallentamiseen löytyi paikallisesta SQLite tietokannasta.

1.1 Lyhenteet ja määritelmät

React Native = avoimeen lähdekoodiin perustuva alustariippumattomien mobiilisovelluksien kehittämiseen tarkoitettu ohjelmistokehys.

Expo = React Nativeen pohjautuva valmis ohjelmistokehityspaketti (Software development kit, SDK).

Redux = yleinen tilanhallintakirjasto Javascript ohjelmointikielelle.

Redux store = sovelluksen globaaleja tiloja hallinnoiva tietovarasto.

SQLite = Relaatiopohjainen tietokanta, jonka sisältö tallennetaan paikalliseen laitteeseen.

NoSql = Not only SQL (engl.). Tietokantamalli, joka ei pohjaudu perinteisiin relaatiotyyppisiin tietokantoihin.

Kirjasto, komponenttikirjasto = useimmiten avoimen lähdekoodin ja kolmannen osapuolen tarjoama modulaarinen kokoelma, lisäosa tai aliohjelma, jonka valmiiksi määritellyt toiminnot on mahdollista lisätä sovellukseen.

Id-avain = Tietokannan yksittäisen dokumentin tai taulurivin yksilöllinen tunniste.

2 KÄYTETTÄVÄT TEKNIIKAT

Nykypäivän alustariippumaton mobiilisovelluskehitys on kehittynyt huimasti verrattuna muutaman vuoden takaiseen tilanteeseen. Oman alkukartoitukseni mukaan ainoat varteenotettavat tekniikat tämän sovelluksen kehittämiseen projektia aloitettaessa olivat React Native ja Flutter. Flutter näyttäytyi omaan arviooni perustuen hieman keskeneräisemmältä verrattuna React Nativeen johtuen kyseessä olevan hieman uudempi tekniikka. Toinen syy React Nativen valintaan oli siihen pohjautuva, itselleni tuttu kehitysympäristö Expo, jolloin uuden kehitysympäristön opettelemiseen ei kuluisi ylimääräistä aikaa.

On mielenkiintoista nähdä, tuleeko Flutter tai React Native vai kenties joku muu vielä julkaisematon tekniikka olemaan hallitsevin tekijä alustariippumattomassa mobiilisovelluskehityksessä lähivuosina.

Seuraavien kappaleiden lisäksi sovelluksessa käytetään monia tässä kappaleessa mainitsemattomia, ulkoasua parantavia ja sovelluksen navigointia helpottavia kirjastoja.

2.1 React Native

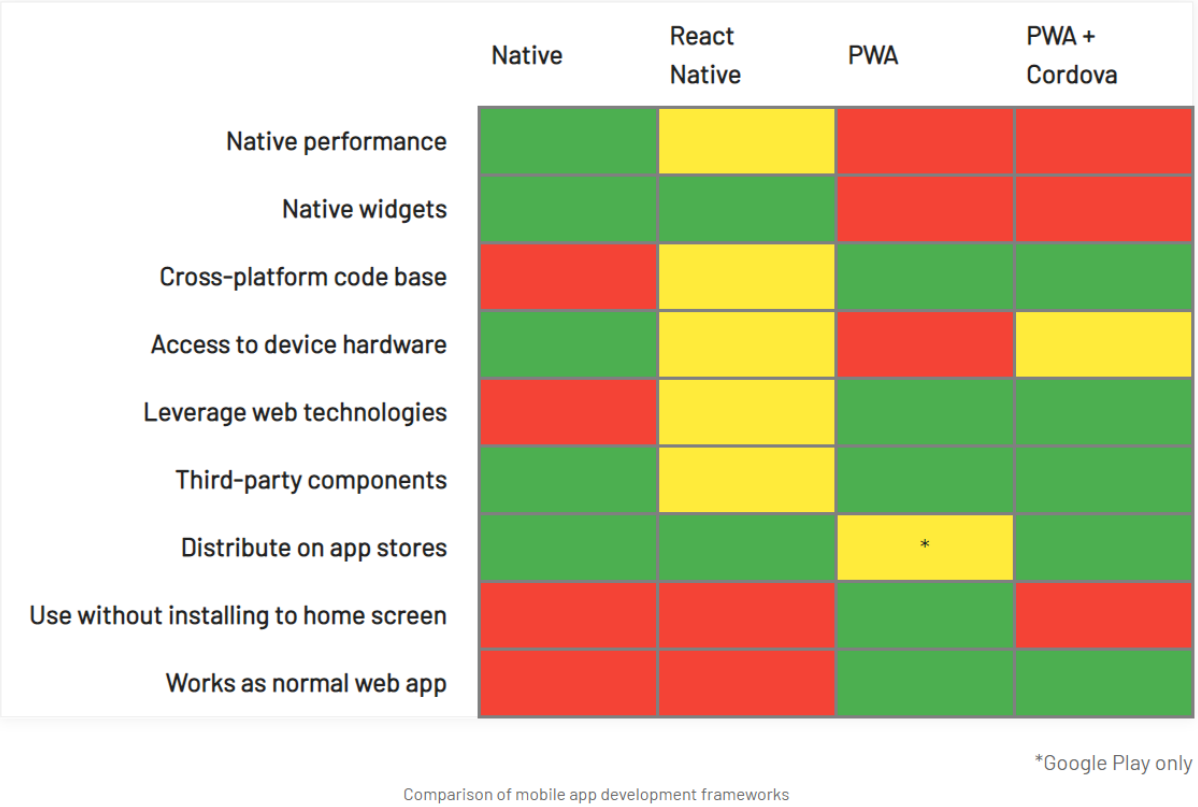
Sovellus kehitettiin Facebookin luomalla React Nativella, joka on avoimeen lähdekoodiin pohjautuva ohjelmistokehys alustariippumattomien mobiilisovellusten kehittämiseen. React Native perustuu React -nimiseen (myös React.js tai ReactJS) JavaScript-kirjastoon, joka on tarkoitettu nopeiden ja tehokkaiden web-käyttöliittymien kehittämiseen. React Nativen vahvuus on yhden ja saman koodin suorittaminen sekä Androidin että iOS:n natiivikomponenttien ymmärtämässä muodossa, jolloin sovellukset näyttävät ja toimivat aivan kuin natiivisovellukset sekä Android että iOS-laitteissa. (React Native 2020)

Vaihtoehtoja React Nativelle ovat natiivi sovelluskehitys ja PWA (Progressive Web Application), eli progressiivinen verkkosovellus.

Natiivi sovelluskehitys tarkoittaa kehittämisen keskittämistä yhdelle käyttöjärjestelmälle kerrallaan, kyseisen käyttöjärjestelmän tukevaa ohjelmointikieltä käyttäen. Nämä ohjelmointikielet ovat Java/Kotlin Androidille sekä Objective C/Swift iOS:lle. Yhteen käyttöjärjestelmään keskityttyessä natiivi sovelluskehitys on lopputulokseltaan useimmissa tapauksissa paras vaihtoehto. Sen huono puoli on luontaisesti korkeat kehitys- ja ylläpito kustannukset, jos vaatimuksena on tehdä sovellus sekä Android, että iOS-käyttöjärjestelmille. (Sysart 2017)

PWA on ikään kuin web-sivuston ja mobiilisovelluksen yhdistelmä, sovelluksesta tallennetaan vain pieni osa käytettävälle laitteelle sovelluksen toimintojen ollessa riippuvaisia verkko- tai mobiiliyhteydestä.

React Native ei varsinaisesti ole hybriditekniikka, mutta ominaisuuksien puolesta sitä voidaan karkeasti pitää natiivi sovelluskehityksen ja PWA:n välimuotona (Kuva 1.).



Kuva 1. React Nativen vertailu natiiviin ja PWA tekniikkaan verrattuna. (Gertner 2019)

React Native on suosituimpien alustariippumattomien mobiilikehitysteknologioiden joukossa. Laajan tuen ansiosta se on erittäin monipuolinen, sillä on kohtalaisen laaja virallinen dokumentaatio ja tietoa löytyy myös muualta internetistä erinomaisesti.
Yksi syy React Nativen valintaan oli myös siihen pohjautuva, itselleni tuttu kehitysympäristö Expo, jolloin uuden kehitysympäristön opettelemiseen ei kuluisi ylimääräistä aikaa.

2.2 Expo

Tarkemmin ottaen sovellus tehtiin Expolla, joka on kehitysympäristöpaketti React Nativelle. Sen suurimmat edut ovat alkuun pääsemisen helppous ja kehittämisen nopea eteneminen sekä iOS, että Android-käyttöjärjestelmälle melkeinpä samaan tahtiin muutamien asetuksien muokkaamista lukuun ottamatta. Suurin etu taitaa kuitenkin olla, että sovellusta voi kehittää Androidin lisäksi myös iOS-käyttöjärjestelmälle ilman macOS-käyttöjärjestelmää, joka ilman Expoa ei onnistu laisinkaan, ellei käytä esimerkiksi virtuaalikonetta, johon on asennettu macOS-käyttöjärjestelmä. Helppouden ja nopeuden lisäksi Expo tuo tiettyjä rajoitteita, kuten mahdollisten käytettävien komponenttikirjastojen vaihtoehdoissa, joka vaikutti myös tähän työhön. Esimerkiksi Bluetooth-toimintoja ei Expossa ole tuettuna tällä hetkellä lainkaan.

On olemassa myös ns. hybridi vaihtoehto, Expon ”ejektointi”, joka vähentää Expon asettamia rajoitteita, mutta samalla poistaa mahdollisuuden tehdä sovellusta iOS:lle Windows-käyttöjärjestelmällä. Toinen ejektoinnin haittapuoli on, että asetuksien määrittämisen helppous molemmille käyttöjärjestelmille poistuu.

Expolle tulee uusia päivityksiä tasaiseen tahtiin poistaen rajoituksia ja tuoden uusia ominaisuuksia. Tällä hetkellä lupaavin tulossa oleva uudistus on Hermes -nimisen mobiili JavaScript-moottorin käyttöönoton mahdollistaminen. Hermes on julkaistu kesällä 2019 Facebookin toimesta React Nativea varten ja tällä hetkellä se tukee virallisesti vain Android sovelluksia Applen asettamien iOS-ohjelmistovaatimuksien vuoksi, mutta tähän saattaa tulla muutos.
Hermes parantaa sovelluksen yleistä suorituskykyä huimasti, vähentää laitteelle asennettavan sovelluksen kokoa huomattavasti ja vähentää myös jonkin verran laitteen keskusmuistin käyttöä.
Expon uusin versio (SDK 37) julkaistiin 1.4.2020, sisältäen React Native version 0.61. Samalla julkistettiin, että jatkossa päivityksiä tulisi neljännesvuosittain. (Facebook Hermes 2019; Expo blog 2020)

2.3 Käytetyt sovellukset

Koodin luomiseen käytettiin Microsoftin kehittämää Visual Studio Codea, monialustaista, avoimeen lähdekoodiin perustuvaa lähdekoodieditoria. Sen etuna on ilmaisuus, toimivuus kaikilla merkittävillä käyttöjärjestelmillä (Windows, Linux, macOS), tuki lukuisille ohjelmointikielille, mahdollisuus laajennuksille mukaan lukien uusien ohjelmointikielien lisääminen sekä sisäänrakennettu tuki Git-versionhallintajärjestelmän komennoille. (Visual Studio Code 2020)

Android Studion emulaattoria käytettiin koodimuutoksien reaaliaikaiseen testaukseen työskentelyn ohessa. Android studio on Googlen ja JetBrainsin kehittämä virallinen kehitysympäristö Android-käyttöjärjestelmälle.

Expo Client on Expon oma mobiilisovellus sekä Androidille että iOS:lle, joka on tarkoitettu koodin reaaliaikaiseen testaamiseen fyysisillä laitteilla. Testaamisprosessista tarkemmin luvussa 7.

2.4 Ohjelmointikielet

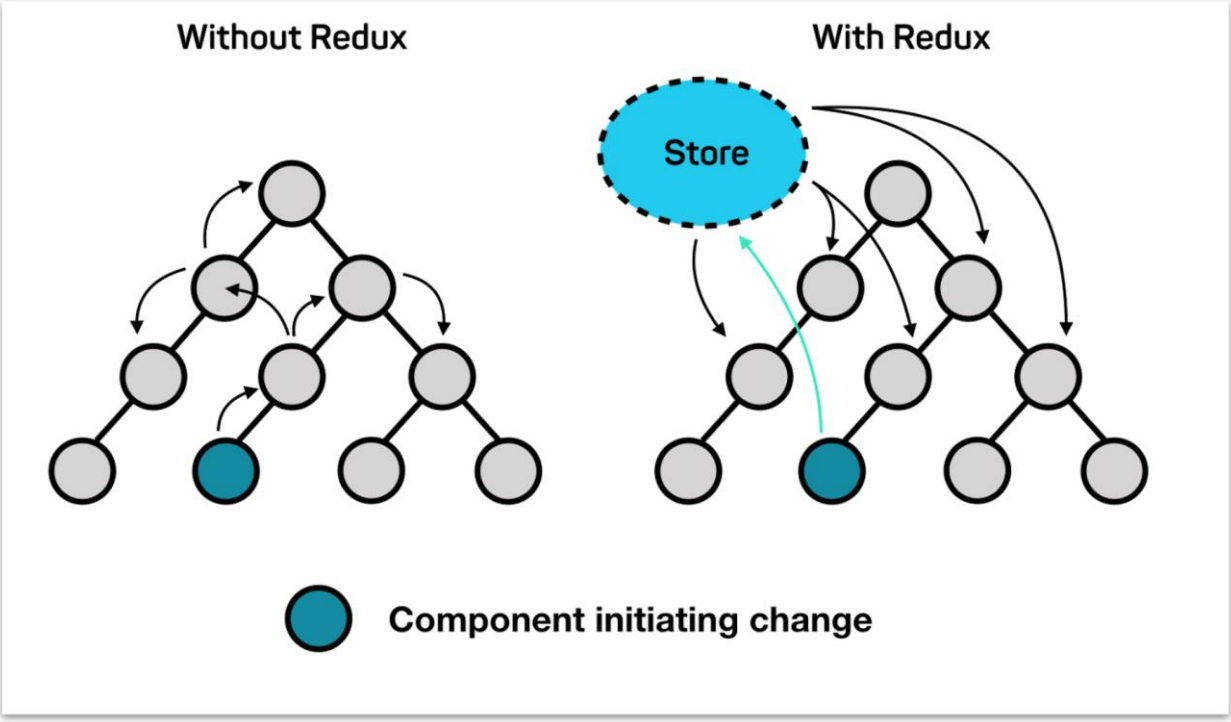
React Nativessa käytettävät ohjelmointikielet ovat JavaScript ja Reactin oma, JSX (JavaScript XML), joka on XML-tyyppinen merkintäkieli. (TIALA 2019, 10)

FireBase-autentikoinnin ja tietokannan hallintaa varten Backend-kielenä toimii JavaScript runtime-ympäristö Node.js.

2.5 Redux

Redux on yleinen kolmannen osapuolen tilanhallintakirjasto React Nativelle. Tilanhallintakirjasto helpottaa sovelluksen käyttämän sisäisen datan hallintaa keskittämällä kaiken muuttuvan datan yhteen, globaaliin paikkaan, jolloin sen päivittäminen ja hakeminen on yksinkertaista esimerkiksi sovelluksen eri välilehdiltä. Tämä globaali tilanhallinta metodi on nimeltään Redux store. Redux store helpottaa sovelluksen eri osien välistä kommunikointia, joka on yleensä välttämätöntä vähänkään isommissa sovelluksissa. (Redux 2020)

Redux storen toimintaidea on esitelty kuvassa 2.



Kuva 2. Reduxin perusidea (Weck 2017.)

2.6 FireBase

FireBase on Googlen tarjoama mobiili- ja web-sovelluksille tarkoitettu pilvipohjainen backend-palvelu. Se sisältää pilvipalveluina 18 eri vaihtoehtoa kuten: NoSQL-tyyppisen tietokannan, tiedostovaraston, sovellusten käyttäjätilien hallinnan, sovelluksen käyttäjien välisen chat-viestittelyn hallinnan ja analytiikka ja diagnostiikka työkaluja sovelluksen kehittämisen avuksi. FireBasella on ilmainen, kuukausimaksullinen (25$/kk) ja käytön määrän perusteella maksettava käyttösuunnitelma. Tämän työn tekemiseen ja testaamiseen riitti ilmainen vaihtoehto. (Google Firebase 2020a)

Tässä työssä käytettiin Cloud Firestore ja Firebase Authentication palveluita. Cloud Firestorea käytettiin reaaliaikaisen sijainnin, kontaktien, julkaistujen sijaintien, chat-keskustelujen ja viestien tallentamiseen sekä hakemiseen. Firebase Authenticationia käytettiin sovelluksen käyttäjätilien luomiseen ja kirjautumisen varmentamiseen.

Alkuperäisenä suunnitelmana oli käyttää Firebase Cloud Messaging palvelua chat-viestinnän hallintaan mutta tällä hetkellä sille ei löydy virallista tai epävirallista tukea Expolle, joten piti turvautua Firestoreen keskustelujen ja viestien tallentamiseen.

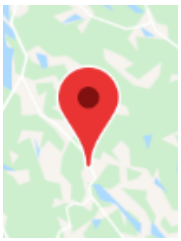
2.7 Komponenttikirjastot

Tässä kappaleessa kerrotaan muista sovelluksessa käytetyistä avoimen lähdekoodin komponenttikirjastoista.

2.7.1 React-native-maps

Tämän työn osalta hyvin oleellinen kirjasto karttanäkymän näyttämiseen ja käsittelemiseen. Vakiona kyseinen kirjasto käyttää Android-käyttöjärjestelmässä Google Mapsia ja iOS:llä Apple Mapsia. Kirjasto nimensä mukaisesti mahdollistaa kartan näyttämisen MapView-komponentilla ja lisäksi lukuisia lisätoimintoja. Kirjasto sisältää myös alikomponentteja käytettäväksi MapView-näkymän sisälle, joista olennaiset tämän työn osalta on: Marker, Circle, Callout, Polygon. (React Native Maps 2020)

Marker on tyypillinen merkki osoittamaan sijainti kartalta (Kuva 3.). Tässä työssä sitä käytettiin näyttämään tallennettu sijaintipiste, toisten käyttäjien sijainti ja ryhmien muodostamisalueen rajan kulmien määrittämisessä.

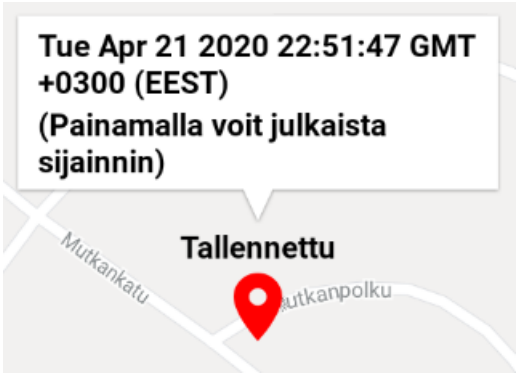


Kuva 3. Merkin oletus ulkoasu.

Circle on ympyränmuotoinen kuvio, joka määritellään keskipisteen ja säteen mukaan (Kuva 4.). Testaamisvaiheessa Circleä käytettiin näyttämään alue, jonka sisältä etsitään muita käyttäjiä.

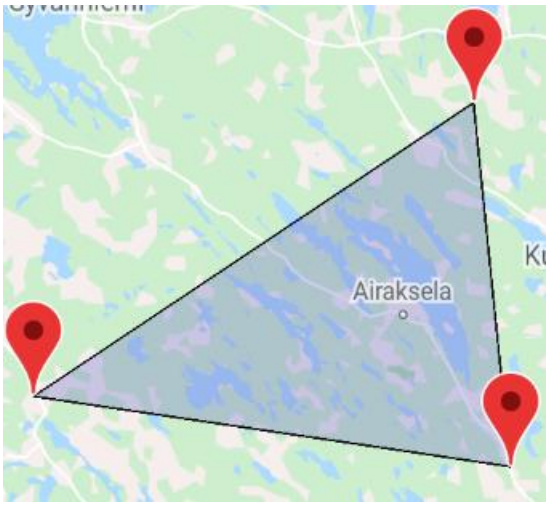
Kuva 4. Circle, jonka keskipiste on käyttäjän sijainti.

Callout on tooltip, jota voi käyttää esimerkiksi merkin infotauluna (Kuva 5.).



Kuva 5. Tooltip näyttää tallennetun sijaintipisteen ajankohdan.

Polygon on monikulmio, jonka kulmat määritetään koordinaatteina. Monikulmio edellyttää vähintään kolme kappaletta kulmapisteitä (Kuva 6.).



Kuva 6. Polygon.

2.7.2 expo-location

Mahdollistaa laitteen sijaintitietojen lukemisen joko kertaluontoisena tai tietyn matkan ja aikavälein. Tarkimmilla mahdollisilla paikannusasetuksilla paikantaminen tapahtuu Wi-Fi:n, matkapuhelinverkon ja GPS:n avulla.

2.7.3 expo-sqlite

SQLite on kevyt, monialustainen ja paikallinen relaatiotietokanta, joka tallentaa tietokannan tiedot käytettävän laitteen yhteen tiedostoon. Tämä on erinomainen tapa tallentaa tietoja, joita ei tarvitse jakaa muille osapuolille. Se tarkoittaa myös hyvää suorituskykyä verrattuna tietokantoihin, jotka tallentavat datan ulkoisille palvelimille. Rajoitteena tai siunauksena riippuen käyttötarkoituksesta on, että tallennettava data on laitekohtaista, eli sitä ei voi hakea suoraan toisilta laitteilta. SQLiten tietoturva on kaksipiippuinen juttu, toisaalta tallennettu data on laitekohtaista, mutta vakiona se ei ole enkryprattya. SQLiteen on olemassa laajennuksia, jotka enkryptoivat datan, tälläinen on esimerkiksi maksullinen SQLite Encryption Extension (SEE). (SQLite 2020a)

SQLite on maailman käytetyin tietokanta, jota käytetään todennäköisesti enemmän kuin kaikkia muita tietokantamoottoreita yhteensä. (SQLite 2020b)

Vuonna 2016 SQLiteä käyttivät muun muassa Google, Facebook, Apple, Adobe sekä monet muut suuret yritykset. (LEPPÄNIEMI 2016, 10)

2.7.4 geolib

Tämä kätevä kirjasto tarjoaa monenlaisia koordinaatteihin perustuvia toimintoja kuten esimerkiksi tämän työn osalta metodit *getPreciseDistance* ja *isPointInPolygon*. Ensimmäisenä mainittu laskee

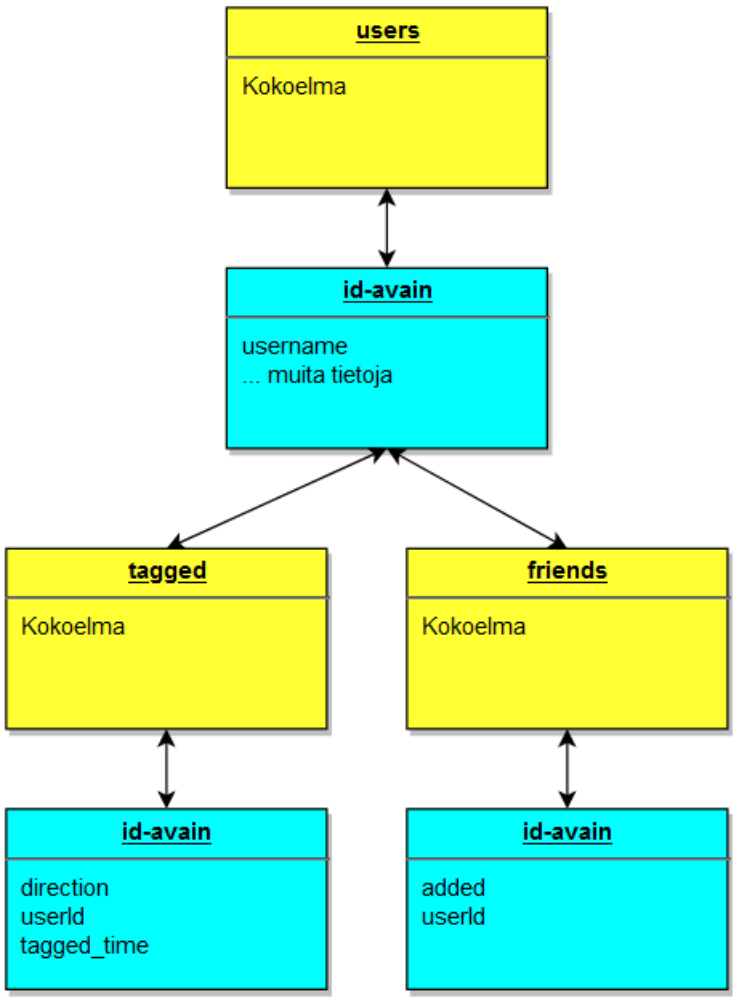
kahden annetun koordinaattipisteen keskinäisen etäisyyden. Jälkimmäiselle metodille annetaan koordinaattitaulukko, joka sisältää monikulmion kulmapisteet ja lisäksi pisteen, jota vasten tarkastetaan, että sisältyykö se annetun monikulmion sisälle. Tätä metodia käytetään sovelluksessa ryhmän luomistoimintoon, jossa tarkastetaan yksi käyttäjää osoittava merkki kerrallaan, että sisältyykö kukin merkki kyseisen monikulmion sisälle.

3 TIETOKANNAN RAKENNE

Firestore Cloud on pilvipalvelupohjainen NoSQL-tietokanta. Tarkennettuna kyseessä on dokumenttipohjainen NoSQL-tietokanta. Käytännössä tämä tarkoittaa, että tietokannan rakenne on kokoelmiin ja dokumentteihin pohjautuva. Tietokannan perusrakenne on muodoltaan: *kokoelma/dokumentti/kokoelma/dokumentti* jne. Yksi kokoelma voi tietenkin sisältää useita dokumentteja, mutta myös dokumentit voivat sisältää useita kokoelmia. Jokainen yksittäinen dokumentti sisältää yksilöllisen tunnisteen dokumentin tunnistamista varten. Lisäksi tallennettavat dokumentit ovat vapaamuotoisia, jolloin niiden ei tarvitse olla dataltaan samanmuotoisia, joka tuo joustavuutta tietojen tallentamiseen. Tässä työssä Firestorea käytetään reaaliaikaisen sijainnin, julkistettujen sijaintien, kavereitten ja ryhmien, käyttäjäkohtaisten tietojen, chat-keskustelujen ja viestien tallentamiseen.

3.1 Käyttäjätunnukset ja kontaktit

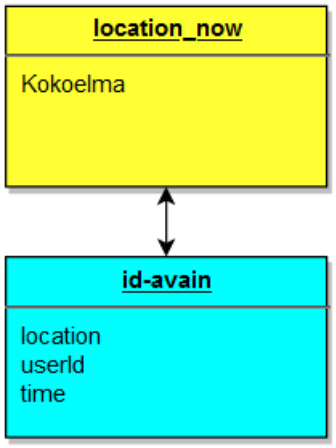
Käyttäjätunnuksia koskeva kansiopolku on tämän työn laajin (Kuva 7.). Jokainen dokumentti sisältää luontaisesti käyttäjäkohtaisen tunnisteen eli id-avaimen, jota käytetään sovelluksen lukuisissa eri toiminnoissa ja tietokantaoperaatioissa, esimerkiksi tunnistamaan kuka käyttäjä tallennetaan ryhmän jäseneksi. Jokaisella käyttäjätunnuksella on tarvittaessa omat kokoelmansa: merkatut henkilöt ja saadut kaveripyynnöt (tagged) ja kaverit (friends).



Kuva 7. Users kokoelman tietokantapolku.

3.2 Sijaintien tallentaminen

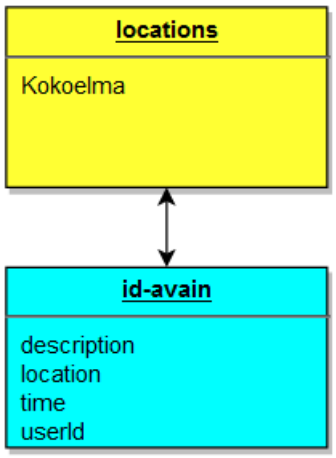
Sijainti välilehden reaaliaikainen sijainti tallennetaan *location_now*-kokoelmaan (Kuva 8.). Jokaiselle tallennetulle dokumentille syötetään id-avaimeksi käyttäjätunnuksen oma tunniste. Tämä tarkoittaa sitä, että käyttäjällä voi olla vain yksi reaaliaikainen sijainti kerrallaan tallennettuna tietokantaan. Tämä helpottaa huomattavasti toisten kohdattujen käyttäjien etsimistä tietokannasta sekä koodin, tietokantahaun suorituskyvyn, että Firebasen tallennuskustannuksien osalta.



Kuva 8. Käyttäjätunnus kohtaiset reaaliaikaiset sijainnit.

Käyttäjän oman laitteen paikallisesta SQLite-tietokannasta on mahdollista julkistaa joku tietty sijaintipiste julkiseksi muille käyttäjille löydettäväksi edellyttäen, että vähintään kahden eri käyttäjätunnuksen julkistetun sijaintipisteen ajan ja matkan etäisyyksien kriteerit täsmäävät.

Kuvassa 9 julkaistujen sijaintien kokoelma.

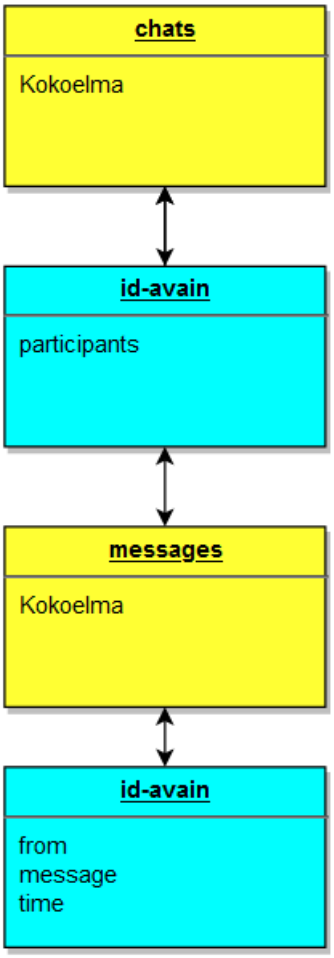


Kuva 9. Julkaistut sijainnit.

3.3 Ryhmät ja viestit

Jokaista kahdenkeskistä chat-keskustelua varten tallennetaan siihen kuuluvat osanottajat yhteen taulukkoon ja jokaista keskustelua kohden voi luonnollisesti olla useampia viestejä, joihin jokaiseen tallennetaan lähettäjä, viestin sisältö ja lähetysaika.

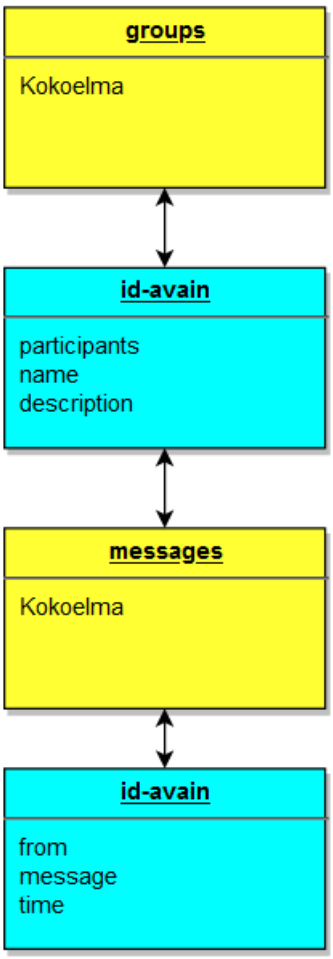
Kuvassa 10 keskustelujen ja viestien polkurakenne.



Kuva 10. Kahden käyttäjän väliset keskustelut ja viestit.

Ryhmiä koskevat chat-keskustelut toimivat täysin samalla periaatteella kuin yllä oleva kahden henkilön välinen keskustelu, ainoana erona on, että osanottajia voi olla enemmän. Lisäksi jokaista ryhmää kohden tallennetaan ryhmän nimi ja kuvausteksti.

Ryhmäkeskustelujen polkurakenteen perusidea on identtinen kahdenkeskisiin keskusteluihin verrattuna (Kuva 11.).



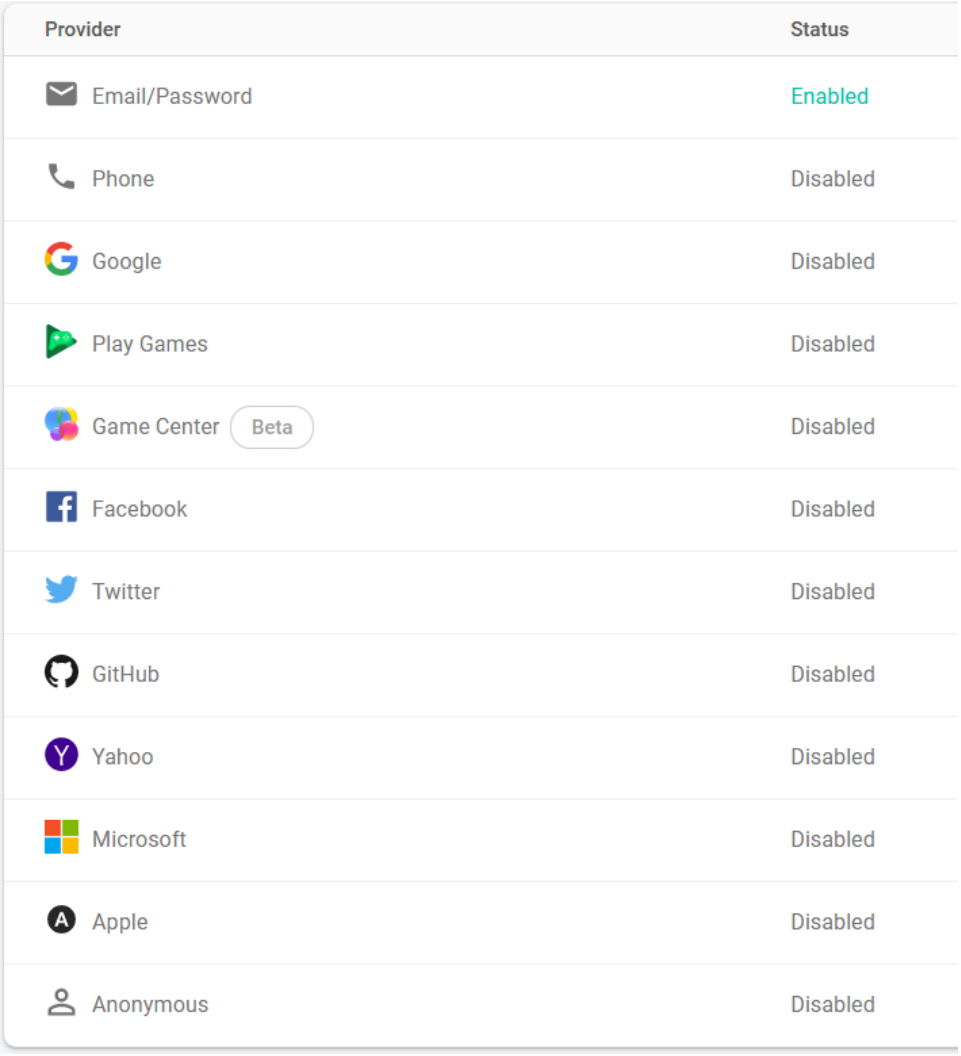
Kuva 11. Ryhmät ja yksittäisen ryhmän keskinäinen keskustelu.

4 SOVELLUKSEN KÄYTTÄMINEN JA TOIMINTOJEN YHTEYS TIETOKANNAN KANSSA

Tässä kappaleessa on mainittu ja kerrottu tarkemmin sovelluksen olennaiset toiminnot käyttäjän kannalta sekä Firebase-tietokannan hyödyntäminen kussakin toiminnossa.

4.1 Käyttäjien hallinta

Firebase Authentication mahdollistaa tunnuksen kirjautumisen tällä hetkellä 12 eri tavalla (Kuva 12.). Näitä ovat mm. Facebook, Twitter ja Google tunnuksien tai puhelinnumeron avulla. Firebase mahdollistaa myös tunnuksen todentamisen mm. puhelinnumeron tai sähköpostin varmistuksella. (Google Firebase 2020b)



Kuva 12. Firebase Authentication tavat.

Tämän työn osalta käyttäjätunnuksien todennus ei ole oleellista, joten tunnuksien luonti ja kirjautuminen tapahtuu sähköposti/salasana -kombinaatiolla myös olemassaolemattomilla sähköpostiosoitteilla. Jokaisella luodulla tunnuksella on oma yksilöllinen id-avain merkkijonona. Tätä id-avainta käytetään mm. kaverisuhteiden, ryhmien ja keskustelujen hallintaan tietokannassa. Tunnusta luodessa käyttäjä antaa sähköpostin ja salasanan lisäksi oman nimimerkin, joka on tunnistettava nimi muille

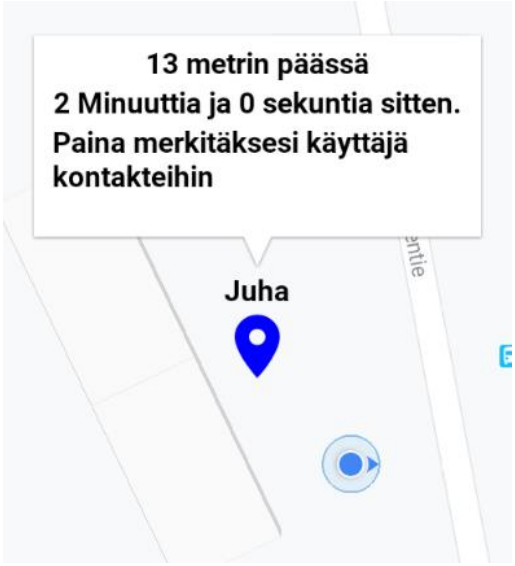
käyttäjille. Myös tämä nimimerkki ja muut tunnusta koskevat lisätiedot tallennetaan tietokannan *users*-kansion dokumenttitiedostoon, jonka nimenä on kyseisen tunnuksen yksilöllinen id-avain. Sovellusta pystyy myös testaamaan vieraana ilman tunnusta, mutta tällöin ainoat mahdolliset toiminnot ovat vain toimintoja, joissa ei käytetä Firebase-tietokantaa. Tämä tarkoittaa käytännössä oman sijainnin paikantamista ja sijaintihistorian tallentamista paikallisesti.

4.2 Käyttäjän oma ja muiden käyttäjien paikantaminen

Sovellusta käyttävän laitteen paikantaminen tapahtuu aikaisemmin mainitulla *expo-location*-metodilla.

Kun käyttäjä kirjautuu sisään sovellukseen, niin samalla aloitussijainti paikannetaan ja tiedot tallennetaan globaaliin Redux storeen nykyisen sijainnin tilaksi. Tämän jälkeen käyttäjän liikkeitä seurataan karttanäkymällä ja tietyn aikavälein tallennetaan käyttäjän sijaintipiste Firebase-tietokannan */location_now*-kansioon dokumentiksi. Dokumentti sisältää kyseisen sijainnin koordinaatit ja tallennusajankohdan. Tärkeä yksityiskohta on, että dokumentin tunnistenimeksi annetaan käyttäjätunnuksen yksilöllinen id-avain, jolloin tuoreita sijainteja on mahdollista olla maksimissaan vain yksi jokaista käyttäjätunnusta kohden. Tämän takia muut käyttäjätunnukset ovat helposti paikannettavissa ja se säästää myös Firebasen tallennuskustannuksissa.

Joka kerta kun oman sijainnin koordinaatit ja aika on tallennettu, niin sen jälkeen haetaan tietokannan samasta kansiosta muita käyttäjiä omaan tallennettuun sijaintiin verrattuna kriteereillä: ajan viive minuutteina ja matkan etäisyys metreinä. Esimerkiksi testaamisen aikana sopivat kriteerit olivat 60 minuutin viive ja 500 metrin etäisyys. Jos nämä molemmat kriteerit toteutuvat niin karttanäkymään ilmestyy merkki, joka näyttää toisen käyttäjän tallennetun sijainnin, nimimerkin ja viiveen verrattuna oman sijaintipisteen tallennettuun aikaan (Kuva 13.). Merkin tooltip ikkunaa painamalla voi merkitä kyseisen käyttäjän, jolloin merkintä tallennetaan tietokantaan ja kommunikointi jatkuu kontaktit-välilehdessä.



Kuva 13. Toinen käyttäjä kohdattu.

Automaattisen nykysijainnin tallentamisen ja muiden käyttäjien etsimisen suoritusväli sekunteina on tällä hetkellä vain koodista käsin määriteltävissä. Tämä asetus olisi helppo lisätä myös sovelluskäyttäjän määriteltäväksi mutta tällöin ongelmaksi saattaa muodostua liian pienestä aikavälistä johtuvat suorituskykyongelmat ja Firebaseen kohdistuvat tiedonsiirtokustannukset.

4.3 Sijaintihistoria

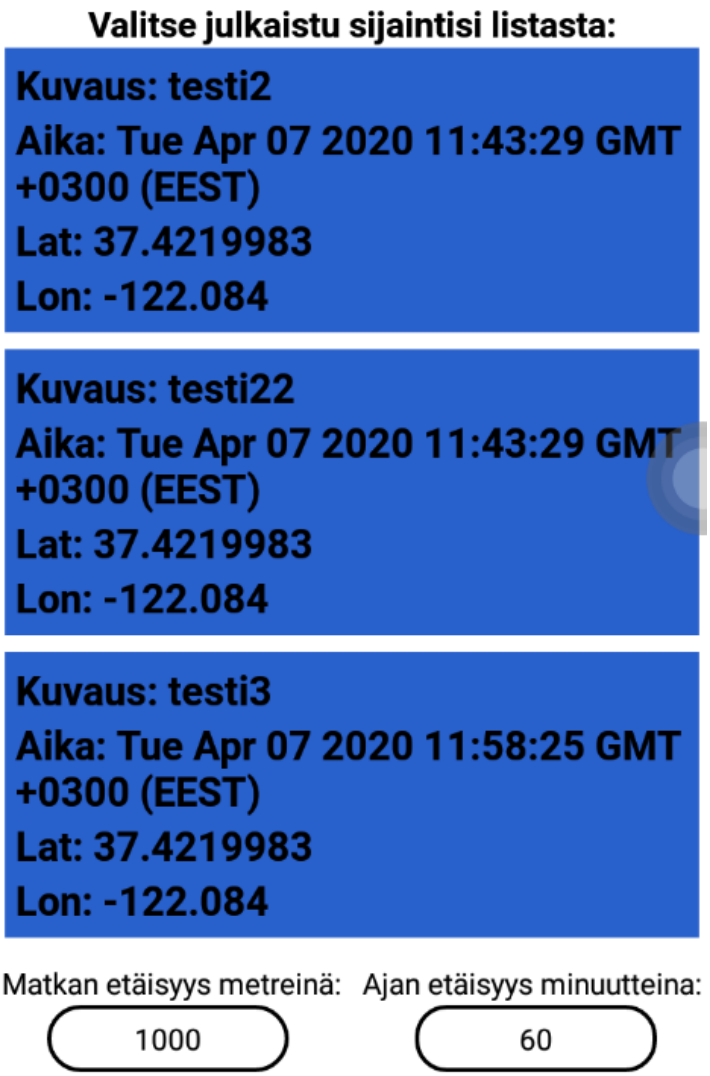
Koska Firebaseen liittyvien siirtokustannuksien takia kokonaista sijaintihistoriaa ei kannata tallentaa Firestore-tietokantaan, niin sopivaksi ratkaisuksi osoittautui paikalliseen laitteeseen tallennettava sijaintihistoria SQLite-tietokantaan. Tietokanta sisältää vain yhden taulun, *locations*, johon tallennetaan sijainnin koordinaatit ja tallentamisajankohta. Yksittäisen sijaintipisteen (Kuva 14.) tallentaminen on mahdollista kertaluontoisesti tai käyttäjän halutessa automaattisesti tietyn sekuntimäärän välein.

Kuva 14. Tallennettu sijaintipiste 2/9 valittu.

Oma sijaintihistoria on näin ollen yksityinen, mutta käyttäjä voi julkaista oman tietyn yksittäisen sijaintipisteensä julkiseksi, jonka perusteella voidaan etsiä toisten käyttäjien julkistettuja sijainteja. Julkaistavalle sijaintipisteelle voi antaa sijaintia tai tapahtumaa kuvaavan nimen esimerkiksi ”Kuopiorock 2021, perjantai ilta”.

Toisten käyttäjien julkistettujen sijaintien etsiminen toimii lähes samalla tavalla kuin reaaliaikaisten sijaintien vertailu, mutta koska julkistetun sijaintipisteen ajankohta ei tarvitse olla reaaliaikainen, niin

tässä tapauksessa ajan etäisyys toimii luonnollisesti molempiin suuntiin, aikaisempaan tai myöhempään. Käyttäjä voi itse määrittää hakuukriteerien ajan ja matkan etäisyydet minuutteina ja metreinä verrattuina muiden käyttäjien julkaisemiin sijaintipisteisiin (Kuva 15).



Kuva 15. Kohtaamisten etsiminen oman julkaistun sijainnin perusteella.

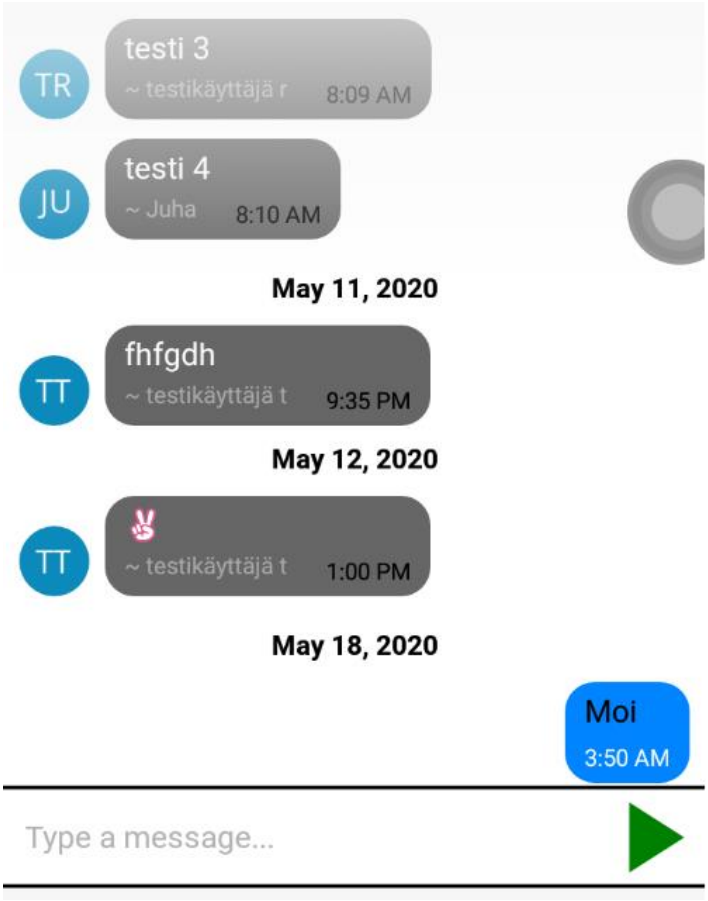
Kun hakukriteereillä löytyy toisen käyttäjän julkaisema sijainti niin samoin kuin reaaliaikaisen kohtaamisen kohdalla, on mahdollista merkitä kohtaaminen, jolloin se tallentuu tietokantaan Kontaktit-välilehteä varten.

Historia-välilehdeltä on myös mahdollisuus tyhjentää oma paikallinen sijaintihistoria.

4.4 Kommunikointi toisten käyttäjien kanssa

Toisten käyttäjien kanssa kommunikointi tapahtuu Kontaktit-välilehdellä. Kaikki kommunikointi kontaktien kanssa toimii teknisesti pelkästään Firestore-tietokannasta käsin. Syynä on Expon tuoma rajoite chat-viestittelyn osalta, koska *react-native-firebase*-kirjasto, joka mahdollistaa Firebase Cloud Messagen käytön React Nativelle, ei ole tuettuna Expolle. Näin ollen piti turvautua Firestore-tietokantaan myös keskustelujen ja viestien tallentamisessa sekä hakemisessa.

Chat-viestittely toimii sekä lisättyjen kaverien että ryhmien kesken (Kuva 16.).



Kuva 16. Testiryhmän chat-keskustelu.

Reaaliaikaisen sijainnin tai julkaistun sijaintihistorian perusteella aluksi kohdatut ja sen jälkeen merkityt käyttäjätunnukset löytyvät merkatut näkymästä (Kuva 17.).

Kuva 17. Kaveripyyntöjen lähettäminen.

Tästä näkymästä (Kuva 17.) on mahdollista lähettää kaveripyyntöjä aikaisemmin merkityille käyttäjätunnuksille. Kun toinen käyttäjä on hyväksynyt kaveripyynnön, molemmat käyttäjätunnukset lisätään toistensa kavereiksi tietokantaan kansioon *users/tunnusavain/friends*.

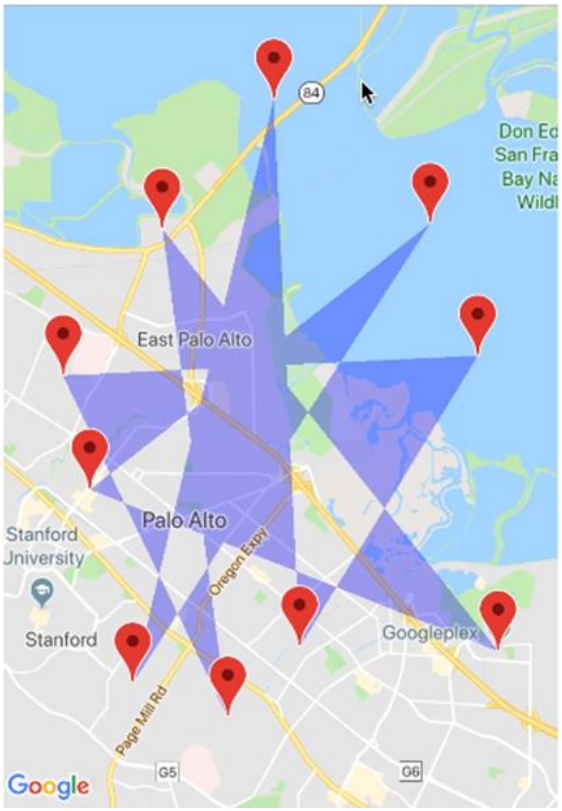
Kaverit ja kaveripyynnöt näkyvät samassa listassa (Kuva 18.).

Kuva 18. Kaverilista.

4.5 Ryhmien luominen

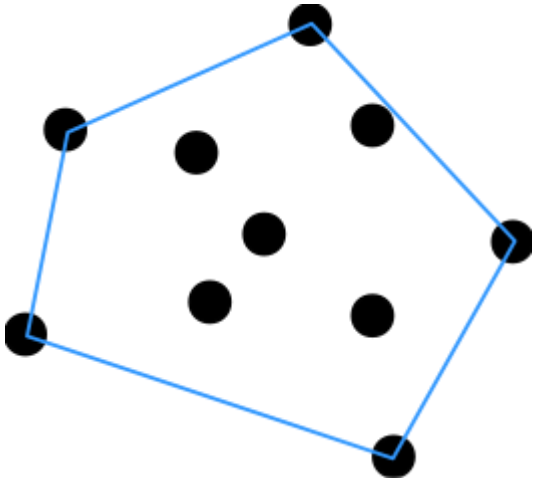
Sijainti-välilehdellä käyttäjä voi reaaliaikaisten kohtaamisten lisäksi myös luoda uusia ryhmiä. Ryhmän luonti tapahtuu kartalle vapaasti muodostettavan monikulmion avulla. Käyttäjä määrittää vähintään kolme pistettä kartalle monikulmioiden kulmapisteiksi, joista muodostetaan valmis kuvio. Uusien ryhmien jäsenet määräytyvät kuvion sisäpuolelle jäävistä käyttäjäsijainneista.

Aluksi ongelmana monikulmion luomisessa oli React Native Mapsin Polygon-komponentin renderöiminen. Kolmen kulmapisteen määritteleminen ei vielä ollut ongelma mutta joissakin tapauksissa jo neljännen kulmapisteen määrittäminen toteutti monikulmion, joka ei ollut ehjä (Kuva 19.).



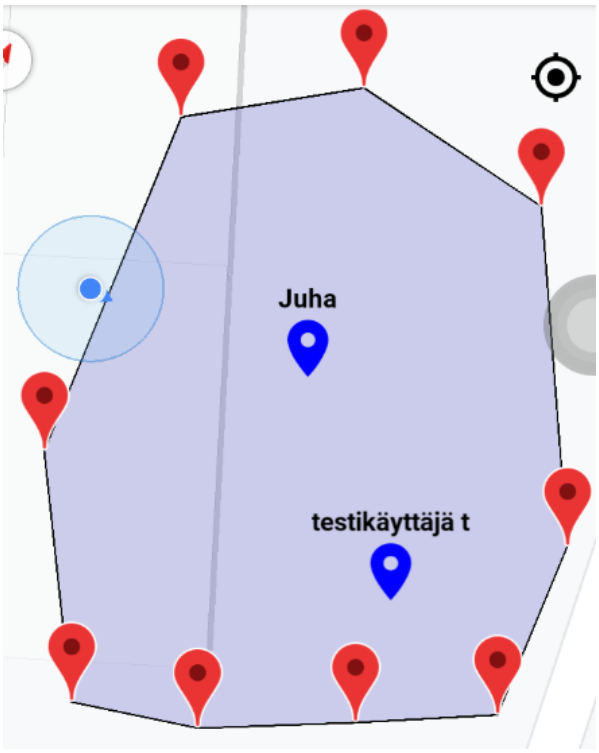
Kuva 19. Monikulmion renderöimisongelma (Kallergis 2019.)

Tekniseksi ratkaisuksi tähän ongelmaan löytyi monikulmion luominen Graham Scan algoritmin avulla. Tämän geometrisen algoritmin julkisti amerikkalainen matemaatikko Ronald Graham vuonna 1972. Algoritmin ideana on etsiä xy-koordinaatistolle merkittyjen pisteiden ulkorajoja määrittävät pisteet eli konveksi runko (convex hull) (Kuva 20.). Algoritmi soveltuu monikulmion piirtämiseen suoraviivaisesti: algoritmille syötetään kaikki käyttäjän määrittelemät kulmapisteet, algoritmin suorittamisen jälkeen jäljelle jää vain pisteet, jotka määrittelevät kaikkien alkuperäisten pisteiden ulkorajoja. Nämä algoritmin palauttamat pisteet käyvät suoraan MapView-karttanäkymän Polygon-komponentille sen piirtämiseksi.



Kuva 20. Esimerkki algoritmin muodostamasta konveksi rungosta. (Wikipedia 2016)

Monikulmion luomisen jälkeen (Kuva 21.) käyttäjä voi siepata kuvion sisälle jäävät käyttäjien sijainteja kuvaavat merkit ryhmäksi, jonka jäseniksi lisätään kyseisten käyttäjien lisäksi ryhmän luoja itse. Ryhmälle annetaan nimi ja kuvaus, joka sen jälkeen tallennetaan Firestore-tietokantaan.



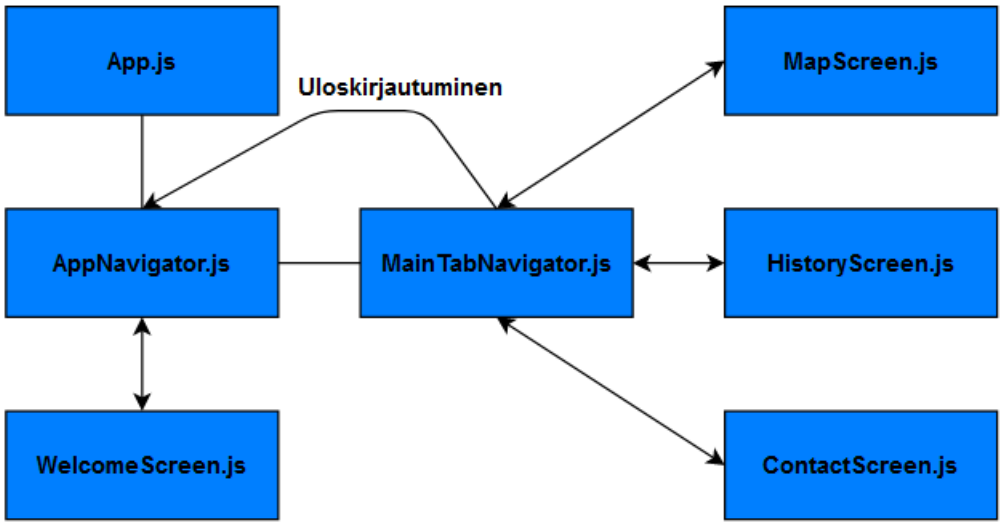
Kuva 21. Ryhmään lisättävien jäsenten määrittävä monikulmio.

5 SOVELLUSNAVIGAATION ARKKITEHTUURI

Sovelluksen navigointiin käytetään apuna React Navigationia, joka on ikään kuin kokoelma navigointitoimintoihin liittyviä komponenttikirjastoja. Sen avulla hallinnoidaan näkymiä, eli sitä mitä ruudulla näkyy kulloinkin.

Kun käyttäjä avaa sovelluksen, niin ensimmäisenä näytetään sovelluksen aloitusruutu, josta käyttäjä voi edetä kirjautumalla sisään. Kirjautumistiedot tarkastetaan Firebase Authenticationin avulla, jonka jälkeen siirrytään ennalta määritellylle välilehdelle, joka on tässä tapauksessa MapScreen-välilehti.

Sovelluksen *App.js*-juuritiedostossa on sovelluksen näkymänä ainoastaan AppNavigator, jonka avulla hallinnoidaan alinäkymiä joko aloitusruutuna tai jonain tiettynä välilehtenä. *AppNavigator.js*-tiedosto sisältää *createSwitchNavigator* -kirjaston, jonka avulla varsinainen välilehtien välinen navigointi sekä uloskirjautuminen tapahtuu. AppNavigator toimii *MainTabNavigator.js*-tiedoston avulla, johon on konfiguroitu sovelluksen kaikkien välilehtien navigointiasetukset *createStackNavigator* -kirjaston avulla. Navigaation arkkitehtuuri näkyy kuvassa 22.



Kuva 22. Navigaation toimintaperiaate.

6 OLENNAISET TEKNISET TOTEUTUKSET

Tässä kappaleessa on kuvattu sovelluksen olennaisia teknisiä toteutuksia. Mainitsematta jää esimerkiksi SQLiten ja Firebasen toiminnot, joiden toteutumiskeinot selviävät niiden virallisesta dokumentaariosta.

6.1 Sijaintien vertaileminen

Sijaintien vertailemiseen käytettiin geolib-komponenttikirjastoa (Kuva 23.), jonka metodit käyttävät geometrisiä algoritmeja koordinaatteihin perustuvien laskelmien saavuttamiseksi.

```
import { getPreciseDistance, isPointInPolygon  } from 'geolib';
```

Kuva 23. geolib-kirjaston metodien tuonti.

Julkaistujen sijaintipisteiden sekä kahden käyttäjän välisen etäisyyden laskeminen tapahtuu *getPreciseDistance*-metodin avulla (Kuva 24.). Funktiota kutsutaan argumenttina taulukko, joka sisältää kaksi sijaintia koordinaatteina, palauttaen koordinaattien välisen etäisyyden metreinä.

```
_getDistance = (locations) => {

  var pdis = getPreciseDistance(
    { latitude: locations[0].latitude, longitude: locations[0].longitude },
    { latitude: locations[1].latitude, longitude: locations[1].longitude }
  );

  return pdis;
};
```

Kuva 24. Kahden sijainnin välisen etäisyyden laskeminen.

Kun käyttäjä luo karttanäkymässä monikulmion ryhmän luomista varten, pitää tarkistaa, että ketkä käyttäjistä sisältyvät monikulmion rajojen sisäpuolelle. Tätä varten tarvitaan *isPointInPolygon*-metodia (Kuva 25.). Metodia kutsutaan argumentteina objekti, joka sisältää käyttäjätunnuksen sijaintikoordinaatit sekä taulukon, joka sisältää monikulmion kulmapisteiden koordinaatit, palauttaen arvoksi tosi tai epätosi riippuen siitä, että sisältyikö käyttäjätunnuksen sijainti monikulmion sisälle.

```
isIn = isPointInPolygon({ latitude: item.location._lat, longitude: item.location._long }, coordinates);
```

Kuva 25.

6.2 Redux

Reduxia käytetään tässä työssä välilehtien keskinäiseen kommunikointiin. Esimerkiksi käyttäjätunnusta koskevat tiedot haetaan kerran käyttäjän kirjautuessa sisään, jolloin tiedot ovat heti käytettävissä jokaisella välilehdellä, eikä tietoja tarvitse hakea Firebasesta eri välilehdiltä erikseen. Tämä välilehtien välinen kommunikointi tapahtuu *app-redux.js*-tiedostoon määriteltyihin globaaleihin tiloihin ja niitä koskeviin toimintoihin.

Seuraavaksi esitellyt funktiot tulee sisällyttää jokaiselle välilehdelle erikseen, jotta globaalit tilat ja toiminnot ovat käytettävissä. Välilehtikohtaiset redux toiminnot näytetty kuvassa 26.

mapStateToProps-funktio pitää yllä *app-redux.js*- tiedostoon määriteltyjä globaaleja tiloja. *mapDispatchToProps* määrittää toimintofunktiot perustuen *app-redux.js*-tiedostoon määriteltyihin toimintoihin, joilla globaaleja tiloja voidaan päivittää.

```
import { getUserData, getDeviceInfo, clearTableData, addMeeting } from "./../redux/app-redux";

const mapStateToProps = (state) => {
  return {
    userData: state.userData,
    device: state.device,
    location_now: state.location_now,
    meetings: state.meetings,
  };
}

const mapDispatchToProps = (dispatch) => {
  return {
    getUserData: () => { dispatch(getUserData()) },
    getDeviceInfo: () => { dispatch(getDeviceInfo()) },
    clearTableData: () => { dispatch(clearTableData()) },
    addMeeting: (meeting) => { dispatch(addMeeting(meeting)) },

  };
}
```

Kuva 26. Redux toimintojen määritteleminen yksittäisellä välilehdellä.

6.3 Graham Scan algoritmin hyödyntäminen

Graham Scan algoritmiä käytetään käyttäjän valitsemien kulmapisteiden ulkorajojen määrittämisessä ehjän monikulmion renderöimiseksi.

Uuden algoritmi instanssin määrittämisen jälkeen lisätään kaikki käyttäjän antamat kulmapisteet *addPoint*-metodilla. Sen jälkeen palautetaan kaikkien pisteiden ulkorajoja määrittävät pisteet taulukkoon käyttämällä *getHull*-metodia. Prosessi näkyy Kuvassa 27 kokonaisuudessaan.

```
const convexHull = new ConvexHullGrahamScan();
var x = this.state.coordinates2.length;

for (var i = 0; i < x; i++)
{
  convexHull.addPoint(this.state.coordinates2[i].longitude, this.state.coordinates2[i].latitude);
}
```

Kuva 27. Algoritmin suorittaminen.

7 TESTAAMINEN

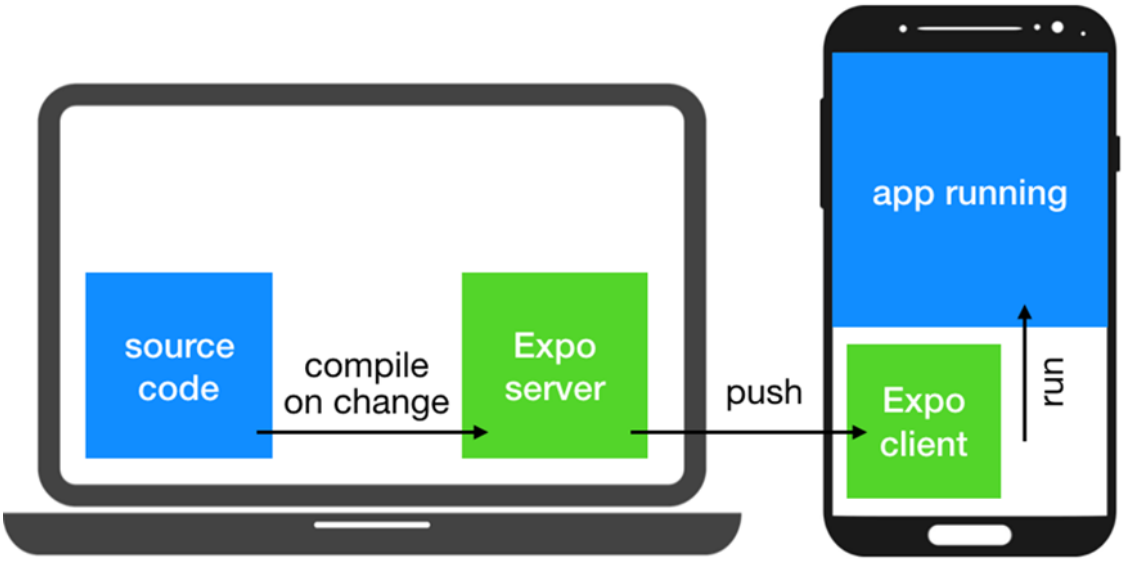
Kehittämisen ohessa testaaminen tapahtui Android Studion emulaattorilla ja omalla Android puhelimalla. iOS-emulaattoria ei ole mahdollista saada toimimaan Windows-käyttöjärjestelmillä ainakaan suoraan, mutta tähän löytyi osittaiseksi ratkaisuksi Expon oma web-pohjainen snack.expo.io, joka mahdollistaa Android- ja iOS-emulaattorien ajamisen internet selaimessa. Snack.expo.io on tarkoitettu lähinnä pienimuotoisten sovelluksien testaamiseen ja itse en saanut esim. Firebasen toimintoja lainkaan toimimaan kyseisellä emulaattorilla enkä löytänyt tähän ratkaisua mistään. Mutta tärkeintä on, että testaaminen onnistui iOS:n ulkoasun oikeellisuuden varmistamiseksi.

Sovelluksen ominaisuuksien testaamisen tosielämässä oli tarkoitus tapahtua Savonian opistotien kampuksella tietyn luokan avustuksella 14.4.2020, mutta vuoden 2019 lopulla alkaneen koronaviruspandemian aiheuttamat kokoontumisrajoitukset estivät testaamisen suuremmalla ryhmällä.

Sovelluksen kommentoimisessa, ohjeistamisessa ja testaamisessa fyysisillä laitteilla virheiden löytämiseksi auttoi toimeksiantaja iOS ja Android -älypuhelimilla. Testaamisessa auttoi myös se, että periaatteessa sovelluksen kaikkia toimintoja voi testata yhdelläkin laitteella, kun välillä kirjautuu sisään toisella tunnuksella.

Expo Client -mobiilisovellusta käytettiin testaamaan koodia fyysisillä laitteilla sekä Androidilla, että toimeksiantajan avustuksella myöskin iOS-älypuhelimella. Expo Clientin huono puoli on sen heikko suorituskyky johtuen sovelluksen suorittamisesta Expon oman palvelimen kautta. Tämä korostuu Firebasen toimintoja käytettäessä, jolloin esimerkiksi tietokantaan kohdistuvat haut menevät Expo-palvelimen kautta, jolloin haun viive on karkeasti noin kaksinkertainen.

Kuvassa 28 on kuvattu testaamisvaiheen toimintamekanismi lähdekoodista suoritettavan Expo Clientin kautta ajettavaan sovellukseen.



Kuva 28. Testaamisvaiheen toimintamekanismi (Maksimović 2018.)

8 YHTEENVETO JA POHDINTA

Ennen opinnäytetyötä React Nativen ja Expon perusteet olivat minulle tuttuja. Työn alkaessa näkemys oli, että React Native on hieman poikkeava muista aikaisemmin käyttämistäni tekniikoista ja näin ollen alkuun on hankalahko päästä mutta alun jälkeen kehittäminen on hyvinkin yksinkertaista ja suoraviivaista. Tämä näkemys laajeni tätä työtä tehdessä. Itselleni uuden navigaatiokomponentin käyttö välilehtien hallintaa varten tuntui aluksi haastavalta, koska se vaikeutti sovellusarkkitehtuurin hahmottamista ja rakentamista. Lopuksi laajemmankin sovelluksen kehittäminen tuntui suoraviivaiselta ja itsestään selvältä. React Nativen monipuolisuus ehkä osittain yllätti minut, kolmannen osapuolen komponenttikirjastoja on runsaasti tarjolla jo pelkästään Expolle sen osittaisesta rajoittuneisuudesta huolimatta lähes minkälaisiin tarkoituksiin ja projekteihin tahansa.

Työssä ei ollut mitään yksittäisiä suuria haasteita vaan ne jakautuivat tasaisesti koko työn osalta, samalla tasaiseen tahtiin uutta oppien. Ulkoasun osalta käyttöliittymän rakenteen suunnittelu oli mainitsemisen arvoinen haaste. Johtuen sovelluksen luonteesta, haasteet jakautuivat myös karttanäkymän alikomponenttien ja niiden koordinaattien käsittelyyn.

Opinnäytetyön aihe oli itselleni mielenkiintoinen ja mielekäs. Aihe on sekä luonteeltaan että teknisessä mielessä hyvin ajankohtainen ja tärkeä, mikä tuotti hieman paineita työn alkupuolella, koska vaadittujen ominaisuuksien ja toimintojen toteuttamistavalle ei ollut olemassa valmista kaavaa vaan kaikki tuli luoda itse. Teknisellä tasolla tärkeimmät oppimani asiat olivat karttanäkymän ja sijaintikoordinaattien käsittely.

Lopputuloksena on sovellus, joka sisältää ennalta määritellyt toiminnot. Ulkoasuun liittyvien viimeistelyjen jälkeen sovellus on valmis sen alkuperäistä tarkoitusta, suunniteltuja jatkotoimenpiteitä varten.

LÄHTEET

Expo blog 2020 [viitattu 2020-05-18] Saatavissa: https://blog.expo.io/expo-sdk-37-is-now-available-dd5770f066a6

Facebook Hermes 2019 [viitattu 2020-05-18] Saatavissa: https://engineering.fb.com/android/hermes/

GERTNER, Matthew 2019. React Native, PWAs and the decline of native mobile app development. Kuva [viitattu 2020-05-20] Saatavissa: https://blog.salsitasoft.com/react-native-pwa-mobile-app-development/

Google Firebase 2020a [viitattu 2020-05-17] Saatavissa: https://firebase.google.com/

Google Firebase 2020b [viitattu 2020-05-24] Saatavissa: https://firebase.google.com/docs/auth

KALLERGIS, John A 2019. Google Maps in React Native Part 3: Advanced Usage. Kuva [viitattu 2020-05-20] Saatavissa: https://jakallergis.com/google-maps-in-react-native-part-3-advanced-usage

LEPPÄNIEMI, Eero 2016. SQLite-tietokantaratkaisu C#-pohjaisessa pelissä. Tampereen ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma, Pelituotanto. Opinnäytetyö [viitattu 2020-05-18] Saatavissa: https://www.theseus.fi/bitstream/handle/10024/115839/Leppaniemi_Eero.pdf?sequence=1&isAllowed=y

Maksimović, Darko 2018. Developing Mau King — React Native in Expo vs. Ejected mode and how to get both at the same time. Kuva [viitattu 2020-05-20] Saatavissa: https://medium.com/developing-mau-king/developing-mau-king-react-native-in-expo-vs-ejected-mode-and-how-to-get-both-at-the-same-time-f36e5af607dc

React Native 2020 [viitattu 2020-05-16] Saatavissa: https://reactnative.dev/docs/intro-react-native-components

React Native Maps 2020 [viitattu 2020-05-18] Saatavissa: https://github.com/react-native-community/react-native-maps

Redux 2020 [viitattu 2020-05-16] Saatavissa: https://redux.js.org/introduction/getting-started

SQLite 2020a [viitattu 2020-05-16] Saatavissa: https://www.sqlite.org/onefile.html

SQLite 2020b [viitattu 2020-05-16] Saatavissa: https://www.sqlite.org/mostdeployed.html

Sysart 2017. Natiivi, hybridi vai React Native? Mobiilisovellustyypit vertailussa. [viitattu 2020-05-23] Saatavissa: https://sysart.fi/blog/2017/11/30/natiivi-hybridi-vai-react-native-mobiilisovellustyypit-vertailussa/

TIALA, Oskari 2019. ALUSTARIIPPUMATON MOBIILIKEHITYS REACTNATIVELLA. Vaasan ammattikorkeakoulu. Tietotekniikka. Opinnäytetyö [viitattu 2020-05-18] Saatavissa: https://www.theseus.fi/bitstream/handle/10024/171059/Tiala_Oskari.pdf?sequence=2&isAllowed=y

Visual Studio Code 2020 [viitattu 2020-05-16] Saatavissa: https://code.visualstudio.com/

Weck, Sandy 2017. Developing modern offline apps with ReactJS, Redux and Electron –Part 3 –ReactJS +Redux. Kuva [viitattu 20-05.20] Saatavissa: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/

Wikipedia 2016 [viitattu 2020-05-20] Saatavissa: https://en.wikipedia.org/wiki/File:Convexhull.svg